A Review of Fault Tolerant Checkpointing Protocols for Mobile Computing Systems

Rachit Garg
Singhania University
Dept. of Computer Sc & Engg
Pacheri Bari (Rajasthan), India

Praveen Kumar
Meerut Institute of Engg & Tech.
Dept of Computer Sc. & Engg
Meerut (INDIA)-125005

ABSTRACT
A distributed system is a collection of independent entities that cooperate to solve a problem that cannot be individually solved. A mobile computing system is a distributed system where some of processes are running on mobile hosts (MHs), whose location in the network changes with time. Mobile distributed systems raise new issues such as mobility, low bandwidth of wireless channels, disconnections, limited battery power and lack of reliable stable storage on mobile nodes. This paper addresses the problem of fault tolerant computing in mobile distributed systems. The techniques described are based on checkpointing and roll back recovery.

Keywords
distributed systems, fault tolerance, **c**heckpointing, mobile computing systems, backward error recovery

1. INTRODUCTION
Distributed computing or cluster computing is being used extensively as they are cost-effective and scalable, and are able to meet the demands of high performance computing. With the increase in the number of components there is a increase in the failure probability. To provide fault tolerance it is essential to understand the nature of the faults that occur in these systems. There are mainly two kinds of faults: permanent and transient. Permanent faults are caused by permanent damage to one or more components and transient faults are caused by changes in environmental conditions. Permanent faults can be rectified by repair or replacement of components. Transient faults remain for a short duration of time and are difficult to detect and deal with. Thus becomes necessary to provide fault tolerance particularly for transient failures in distributed computers. Fault-tolerant techniques enable a system to perform tasks in the presence of faults and involves fault detection, fault location, fault containment and fault recovery. Fault Tolerance Techniques enable systems to perform tasks in the presence of faults. The likelihood of faults grows as systems are becoming more complex and applications are requiring more resources, including execution speed, storage capacity and communication bandwidth. Reliability and resilience are critical issues in parallel and distributed systems [8]. These systems comprise of various computing devices and communication and storage resources. There are a number of fault sources in a system, including physical failure of components, environmental interference, software errors, security violations, and operator errors. Faults can be classified into two types: permanent and transient faults. Permanent faults are faults that cause a permanent damage to some part of the system. Recovery from permanent faults must include replacement of the damaged part and reconfiguration of the system. Transient faults are short-lived and do not lead to permanent damage. Recovery from transient faults is comparatively simple as compared to the permanent faults, because reconfiguration of the system is not needed. Generally, the detection of the transient faults is more difficult, because they may disappear without a detectable effect of the system [8].

Fault tolerance can be achieved through some kind of redundancy. Redundancy can be temporal or spatial. In temporal redundancy, i.e., checkpoint-restart, an application is restarted from an earlier checkpoint or recovery point after a fault. This may result in the loss of some processing and applications may not be able to meet strict timing targets. In spatial redundancy, many copies of the application execute on different processors concurrently and strict timing constraints can be met. But the cost of providing fault tolerance using spatial redundancy is quite high and may require extra hardware.

In scientific and commercial applications, in case of a detection of a transient fault, the execution of the program needs to be interrupted and resumed from beginning. As a result, the big applications are completed only if a sufficiently long fault-free interval of time exists in the system. In the presence of faults, the average execution of the program may grow exponentially with the length of the program. Checkpointing is primarily used to avoid losing all the useful processing done before a fault has occurred. Checkpointing consists of intermittently saving the state of a program in a reliable storage medium. Upon detection of a fault, previous consistent state is restored. In case of a fault, checkpointing enables the execution of a program to be resumed from a previous consistent state rather than resuming the execution from the beginning. In this way, the amount of useful processing lost because of the fault is significantly reduced. With checkpointing, the average execution of a program grows only linearly with the length of the program [8].

Checkpoint-Restart or Backward error recovery is quite inexpensive and does not require extra hardware in general. Besides providing fault tolerance, checkpointing can be used for process migration, debugging distributed applications, job swapping, postmortem analysis and stable property detection [95].

There are two approaches for error recovery:

In **forward error recovery techniques**, the nature of errors and damage caused by faults must be completely and accurately assessed and so it becomes possible to remove those errors in the

process state and enable the process to move forward [70]. In distributed system, accurate assessment of all the faults may not be possible.

In **backward error recovery techniques**, the nature of faults need not be predicted and in case of error, the process state is restored to previous error-free state. It is independent of the nature of faults. Thus, backward error recovery is more general recovery mechanism [14], [56].

There are three steps involved in backward-error recovery. These are:

Checkpointing the error-free state periodically, Restoration in case of failure and Restart from the restored state.

Backward error recovery is also known as checkpoint-restore-restart (CRR) or checkpoint-restart (CR). The checkpointing process is executed periodically to advance the recovery line.

2. CHECKPOINTING

A checkpoint is a local state of a process saved on stable storage necessary to allow resumption of processing at a later time. Checkpointing is the process of saving the status information. In a distributed system, since the processes in the system do not share memory, a global state of the system is defined as a set of local states, one from each process. The state of channels corresponding to a global state is the set of messages sent but not yet received. A lost or in-transit message is one, the sending of which has been recorded by the sender but whose receiving could not be recorded by the receiving process. An orphan message is a message whose receive event is recorded, but its send event is lost. A global state is said to be "consistent" if it contains no orphan message and all the in-transit messages are logged. In Figure 1.1, the initial global state {C10, C20, C30, C40, C50} is consistent. It should be noted that initial global state is always consistent, because, it can not contain any orphan message. The Global State {C11, C21, C31, C41, C51} is also consistent, because, it does not possess any orphan message. It needs to be noted that by definition, m0 is not an orphan message but in-transit message. The Global State {C12, C22, C32, C42, C52} is inconsistent because it includes the orphan message m8. By definition, m8 is an orphan message. To recover from a failure, the system restarts its execution from a previous consistent global state saved on the stable storage during fault-free execution. This saves all the computation done up to the last checkpointed state and only the computation done thereafter needs to be redone [8], [77], [78].

After a failure, a system must be restored to a consistent system state. Essentially, a system state is consistent if it could have occurred during the preceding execution of the system from its initial state, regardless of the relative speeds of individual processes. This assumes that the total execution of the system is equivalent to some fault free execution [8]. It has been shown that two local checkpoints being causally unrelated is a necessary but not sufficient condition for them to belong to the same consistent global checkpoint. This problem was first addressed by Netzer and Xu who introduced the notion of a Z-path between local checkpoints to capture both their causal and hidden dependencies [62]. Considering a checkpoint and communication pattern, the rollback dependency trackability property stipulates that there is no hidden dependency between local checkpoints [11]. To be able to recover a system state, all of its individual process states must be able to be restored. A consistent system state in which each process state can be restored is thus called a recoverable system state.

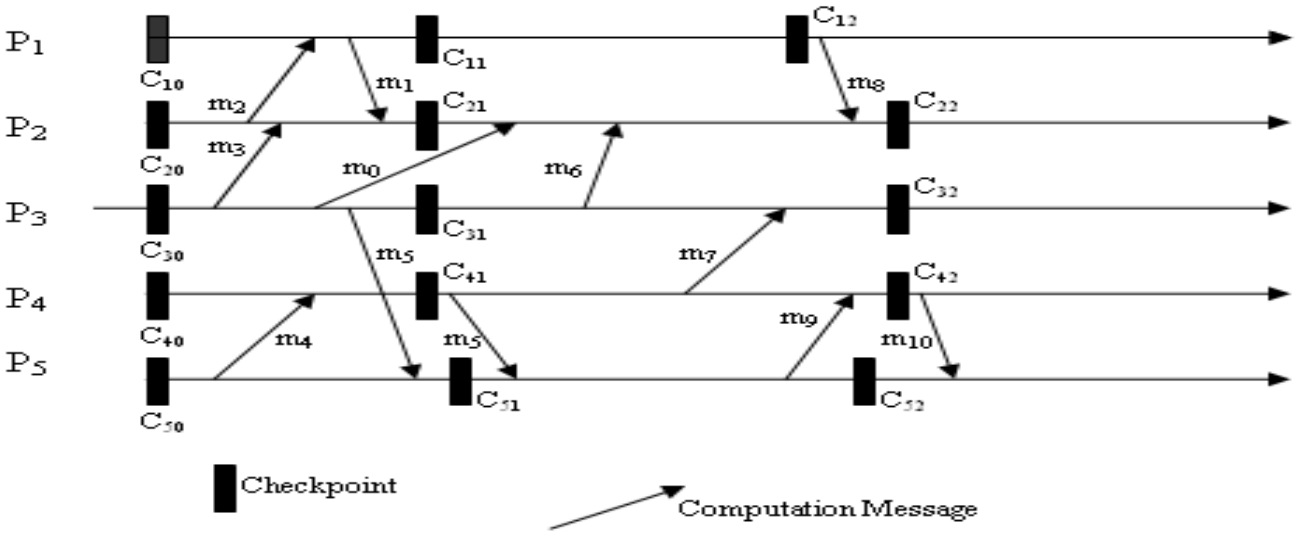

Figure 1.1 Consistent and Inconsistent Global States

Processes in a distributed system communicate by sending and receiving messages. A process can record its own state and messages it sends and receives; it can record nothing else. To determine a global system state, a process Pi must enlist the cooperation of other processes that must record their own local states and send the recorded local states to Pi. All processes cannot record their local states at precisely the same instant

unless they have access to a common clock. We assume that processes do not share clocks or memory. The problem is to devise algorithms by which processes record their own states and the states of communication channels so that the set of process and channel states recorded form a global system state. The global state detection algorithm is to be superimposed on the underlying computation; it must run concurrently with, but not alter, this underlying computation [22].

The state detection algorithm plays the role of a group of photographers observing a panoramic, dynamic scene, such as a sky filled with migrating birds- a scene so vast that it cannot be captured by a single photograph. The photographers must take several snapshots and piece the snapshots together to form a picture of the overall scene. All snapshots cannot be taken at precisely the same instant because of synchronization problems. Furthermore, the photographers should not disturb the process that is being photographed. Yet, the composite picture should be meaningful. The problem before us is to define meaningful and then to determine how the photographs should be taken [22].

The problem of taking a checkpoint in a message passing distributed system is quite complex because any arbitrary set of checkpoints cannot be used for recovery [22], [77], [78]. This is due to the fact that the set of checkpoints used for recovery must form a consistent global state.

In backward error recovery, depending on the programmer's intervention in process of checkpointing, the classification can be:

User-Triggered checkpointing

Transparent Checkpointing

User triggered checkpointing schemes require user interaction and are useful in reducing the stable storage requirement [27]. These are generally employed where the user has the knowledge of the computation being performed and can decide the location of the checkpoints. The main problem is the identification of the checkpoint location by a user.

The transparent checkpointing techniques do not require user interaction and can be classified into following categories:

2.1 Uncoordinated Checkpointing

In uncoordinated or independent checkpointing, processes do not coordinate their checkpointing activity and each process records its local checkpoint independently [14], [86], [96]. It allows each process the maximum autonomy in deciding when to take checkpoint, i.e., each process may take a checkpoint when it is most convenient. It eliminates coordination overhead all together and forms a consistent global state on recovery after a fault [14]. After a failure, a consistent global checkpoint is established by tracking the dependencies. It may require cascaded rollbacks that may lead to the initial state due to domino-effect [44], [77], [78]. It requires multiple checkpoints to be saved for each process and periodically invokes garbage collection algorithm to reclaim the checkpoints that are no longer needed. In this scheme, a process may take a useless checkpoint that will never be a part of global consistent state. Useless checkpoints incur overhead without advancing the recovery line [27].

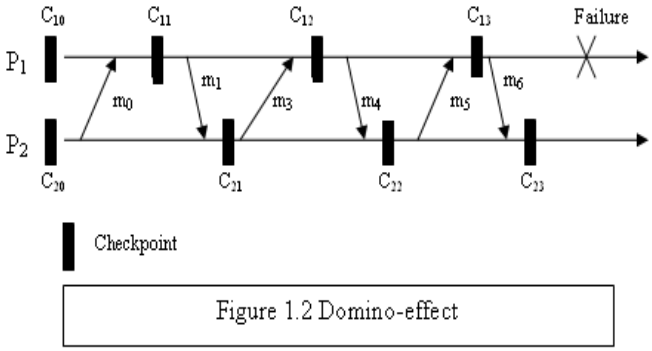

Figure 1.2 Domino-effect

The main disadvantage of this approach is the domino-effect [Figure 1.2]. In this example, processes P1 and P2 have independently taken a sequence of checkpoints. The interleaving of messages and checkpoints leave no consistent set of checkpoints for P1 and P2, except the initial one at {C10, C20). Consequently, after P1 fails, both P1 and P2 must roll back to the beginning of the computation [44]. It should be noted that global state {C11, C21} is inconsistent due to orphan message m1. Similarly, global state {C12, C22} is inconsistent due to orphan message m4.

2.2 Coordinated Checkpointing

In coordinated or synchronous checkpointing, processes take checkpoints in such a manner that the resulting global state is consistent. Mostly it follows two-phase commit structure [22], [28], [44]. In the first phase, processes take tentative checkpoints and in the second phase, these are made permanent. The main advantage is that only one permanent checkpoint and at most one tentative checkpoint is required to be stored. In case of a fault, processes rollback to last checkpointed state. A permanent checkpoint can not be undone. It guarantees that the computation needed to reach the checkpointed state will not be repeated. A tentative checkpoint, however, can be undone or changed to be a permanent checkpoint.

A straightforward approach to coordinated checkpointing is to block communications while the checkpointing protocol executes [88]. A coordinator takes a checkpoint and broadcasts a request message to all processes, asking them to take a checkpoint. When a process receives the message, it stops its executions, flushes all the communication channels, takes a tentative checkpoint, and sends an acknowledgement message back to the coordinator. After the coordinator receives acknowledgements from all processes, it broadcasts a commit message that completes the two-phase checkpoint protocol. On receiving commit, a process converts its tentative checkpoint into permanent one and discards its old permanent checkpoint, if any. The process is then free to resume execution and exchange messages with other processes.

The coordinated checkpointing protocols can be classified into two types: blocking and non-blocking. In blocking algorithms, as mentioned above, some blocking of processes takes place during checkpointing [44], [88]. In non-blocking algorithms, no blocking of processes is required for checkpointing [22], [28]. The coordinated checkpointing algorithms can also be classified into following two categories: minimum-process and all process

algorithms. In all-process coordinated checkpointing algorithms, every process is required to take its checkpoint in an initiation [22], [28]. In minimum-process algorithms, minimum interacting processes are required to take their checkpoints in an initiation [44].

2.3 Quasi-Synchronous or Communication Induced Checkpointing

Communication-induced checkpointing avoids the domino-effect without requiring all checkpoints to be coordinated [12], [33], [55]. In these protocols, processes take two kinds of checkpoints, local and forced. Local checkpoints can be taken independently, while forced checkpoints are taken to guarantee the eventual progress of the recovery line and to minimize useless checkpoints. As opposed to coordinated checkpointing, these protocols do no exchange any special coordination messages to determine when forced checkpoints should be taken. But, they piggyback protocol specific information [generally checkpoint sequence numbers] on each application message; the receiver then uses this information to decide if it should take a forced checkpoint. This decision is based on the receiver determining if past communication and checkpoint patterns can lead to the creation of useless checkpoints; a forced checkpoint is taken to break these patterns [27], [55].

2.4 Message Logging Based Checkpointing Protocols

Message-logging protocols (for example [3], [4], [5], [6], [9], [29], [30], [40], [74], [87], [90], [91], [92], [93], are popular for building systems that can tolerate process crash failures. Message logging and checkpointing can be used to provide fault tolerance in distributed systems in which all inter-process communication is through messages. Each message received by a process is saved in message log on stable storage. No coordination is required between the checkpointing of different processes or between message logging and checkpointing. The execution of each process is assumed to be deterministic between received messages, and all processes are assumed to execute on fail stop processes.

When a process crashes, a new process is created in its place. The new process is given the appropriate recorded local state, and then the logged messages are replayed in the order the process originally received them. All message-logging protocols require that once a crashed process recovers, its state needs to be consistent with the states of the other processes [27], [98]. This consistency requirement is usually expressed in terms of orphan processes, which are surviving processes whose states are inconsistent with the recovered states of crashed processes. Thus, message- logging protocols guarantee that upon recovery, no process is an orphan. This requirement can be enforced either by avoiding the creation of orphans during an execution, as pessimistic protocols do, or by taking appropriate actions during recovery to eliminate all orphans as optimistic protocols do. Bin Yao et al. [98] describes a receiver based message logging protocol for mobile hosts, mobile support stations and home agents in a Mobile IP environment, which guarantees independent recovery. Checkpointing is utilized to limit log size and recovery latency.

3. ASPECTS OF CHECKPOINTING

3.1 Frequency of Checkpointing

A checkpointing algorithm executes in parallel with the underlying computation. Therefore, the overheads introduced due to checkpointing should be minimized. Checkpointing should enable a user to recover quickly and not lose substantial computation in case of an error, which necessitates frequent checkpointing and consequently significant overhead. The number of checkpoints initiated should be such that the cost of information loss due to failure is small and the overhead due to checkpointing is not significant. These depend on the failure probability and the importance of computation. For example, in transaction processing system when every transaction is important and information loss is not permitted, a checkpoint may be taken after every transaction, increasing the checkpoint overhead significantly [42].

3.2 Contents of a Checkpoint

The state of a process has to be saved in stable storage so that the process can be restarted in case of an error. The state/context includes code, data, and stack segments along with the environment and the register contents. Environment has the information about the various files currently in use and the file pointers. In case of message passing systems, environment variables include those messages which are sent and not yet received. The information that is necessary to resume a computation after it is pre-empted is called the context of that computation [42].

3.3 Overheads of Checkpointing Algorithm

During a failure free run, every global checkpoint incurs coordination overhead and context saving overhead in a multiprocessor system. In parallel/distributed systems, coordination among processes is needed to obtain a consistent global state. Special messages and piggybacked information with regular messages are used to obtain coordination among processes. Coordination overhead is due to special control messages and piggybacked information. The book-keeping operations necessary to maintain coordination also contribute to coordination overhead. The time taken to save the global context of a computation is defined as the context saving overhead. If stable storage is not available with every node in a multiprocessor system, the context is transferred over the network. Network transmission delay is also included in the overhead [42].

3.4 Application of Checkpointing

Besides its use to recover from failures, checkpointing is also used in debugging distributed programs and migrating processes in multiprocessor system. In debugging distributed programs, state changes of a process during execution are monitored at various time instances. Checkpoints assist in such monitoring. To balance the load of processors in the distributed system, processes are moved from heavily loaded processors to lightly loaded ones. Checkpointing a process periodically provides the information necessary to move it from one processor to another [42]. With checkpointing, an arbitrary temporal section of a program's runtime can be extracted for exhaustive analysis without the need to restart the program from beginning [26].

3.5 Checkpointing Issues

The existence of mobile nodes in a distributed system introduces new issues that need proper handling while designing a checkpointing algorithm for such systems. These issues are mobility, disconnections, finite power source, vulnerable to physical damage, lack of stable storage etc. [1], [10]. The location of an MH within the network, as represented by its current local MSS, changes with time. Checkpointing schemes that send control messages to MHs, will need to first locate the MH within the network, and thereby incur a search overhead [2]. Due to vulnerability of mobile computers to catastrophic failures, disk storage of an MH is not acceptably stable for storing message logs or local checkpoints. Checkpointing schemes must therefore, rely on an alternative stable repository for an MH's local checkpoint [2]. Disconnections of one or more MHs should not prevent recording the global state of an application executing on MHs. It should be noted that disconnection of an MH is a voluntary operation, and frequent disconnections of MHs is an expected feature of the mobile computing environments [2]. The battery at the MH has limited life. To save energy, the MH can power down individual components during periods of low activity [31]. This strategy is referred to as the doze mode operation. The MH in doze mode is awakened on receiving a message. Therefore, energy conservation and low bandwidth constraints require the checkpointing algorithms to minimize the number of synchronization messages and the number of checkpoints.

The new issues make traditional checkpointing techniques unsuitable to checkpoint mobile distributed systems [1], [20], [57], [72]. Prakash-Singhal [72] proposed that a good checkpointing protocol for mobile distributed systems should have low memory overheads on MHs, low overheads on wireless channels and should avoid awakening of an MH in doze mode operation. The disconnection of MHs should not lead to infinite wait state. The algorithm should be non-intrusive, coordinated, and should force minimum number of processes to take their local checkpoints.

Minimum-process coordinated checkpointing is an attractive approach to introduce fault tolerance in mobile distributed systems transparently. It avoids domino-effect, minimizes stable storage requirements, and forces only minimum interacting processes to checkpoint. To recover from a failure, the system simply restarts its execution from a previous consistent global checkpoint saved on the stable storage. But, it has the following disadvantages. Some blocking of processes takes place or some useless checkpoints are taken. In order to record a consistent global checkpoint, processes must synchronize their checkpointing activities. In other words, when a process initiates checkpointing procedure, it asks all relevant processes to take their checkpoints. Therefore, coordinated checkpointing suffers from high overhead associated with the checkpointing process. Sometimes, checkpoint sequence numbers are piggybacked along with computation messages. If a single process fails to checkpoint, the whole checkpointing effort of the particular initiation goes waste.

Acharya, A. [2] cast distributed systems with mobile hosts into a two tier structure: 1) a network of fixed hosts with more resources in terms of storage, computing, and communication, and 2) mobile hosts, which may operate in a disconnected, or doze mode, connected by a low bandwidth wireless connection to this network. They propose a two tier principle for structuring distributed algorithms for this model:

To the extent possible, computation and communication costs of an algorithm is borne by the static network. The core objective of the algorithm is achieved through a distributed execution amongst the fixed hosts while performing only those operations at the mobile hosts that are necessary for the desired overall functionality.

In wireless cellular network, mobile computing based on a two-tier coordinated checkpointing algorithm reduces the number of synchronization messages [46].

3.6 Related Concepts

When processes interact with each other by exchanging messages, dependency is introduced among the events of different processes, making it difficult to have a total ordering of events. Lamport [52] pointed out this and he proposed a relation called 'happened before' (denoted by $\rightarrow$) to have a partial ordering of events in a distributed system. This is an irreflexive, anti-symmetric, transitive relation.

If a and b are two events occurring in the same process and if a occurs before b, then $a \rightarrow b$. If a is the event of sending a message and b is the event of receiving the same message, then $a \rightarrow b$. Two events a and b are said to be concurrent if and only if a does not happen before b and b does not happen before a. Local checkpoint is an event that records the state of a process at a processor at a given instant. Global checkpoint is a collection of local checkpoints, one from each process. A global state is said to be consistent if all the included events form a concurrent set. A consistent global checkpoint is a collection of local checkpoints, one from each process, such that each local checkpoint is concurrent to every other local checkpoint. Rollback recovery is a process of resuming/recovering a computation from a consistent global checkpoint.

The messages generated by the underlying computation are referred to as computation messages or simply messages and are denoted by m_i or m. The processes are denoted by P_i. The i^{th} CI of a process denotes all the computation performed between its i^{th} and $(i+1)^{th}$ checkpoint, including the i^{th} checkpoint but not the $(i+1)^{th}$ checkpoint.

A process P_i directly depends upon P_j only if there exist m such that: (i) P_i has processed m sent by P_j (ii) P_i has not taken any permanent checkpoint after processing m (iii) P_j has not taken any permanent checkpoint after sending m. Direct dependencies at P_i can be stored in a bit vector of length n for n processes [say $ddv_i[]$]. $ddv_i[j]=1$ implies P_i is directly dependent upon P_j. In minimum-process coordinated checkpointing, if P_i takes its checkpoint and P_i is dependent upon P_j, then P_j should also take its checkpoint. Minimum set is the set of processes which need to checkpoint in an initiation. A process is in the minimum set only if the initiator process is transitively dependent upon it. A process that initiates checkpointing is called initiator process or simply initiator. The minimum-process algorithms are generally based on keeping track of direct

dependencies among processes and computing minimum set [48], [64].

Once the system has rolled back to a consistent state, the nodes have to retrace their computation that was undone during the rollback. The following types of messages have to be handled while retracing the lost computation [72].

Orphan Messages: Messages whose reception has been recorded, but the record of their transmission has been lost. This situation arises when the sender node rolls back to a state prior to sending the message while the receiver node still has the record of its reception.

Lost Messages: Messages whose transmission has been recorded, but the record of their reception has been lost. This happens if the receiver rolls back to a state prior to the reception of the message, while the sender does not roll back to a state prior to their sending.

Duplicate Messages: This happens when more than one copy of the same message arrives at a node; perhaps one corresponding to the original computation and one generated during recovery phase. If the first copy has been processed, all subsequent copies should be discarded.

In deterministic systems, if two processes start in the same state, and both receive the identical sequence of inputs, they will produce the identical sequence outputs and will finish in the same state. The state of a process is thus completely determined by its starting state and by sequence of messages it has received [38], [39].

Chandy-Lamport algorithm [22] works with FIFO channels only. If a message m_1 followed by m_2 is sent from P_i to P_j, m_1 reaches before m_2 when the channels are FIFO. Advantage of a FIFO channel is that without explicitly sending any message sequence numbers with messages, it is possible to arrange the messages in a sequence. Non-FIFO channels necessitate headers with regular messages to ensure correct ordering of messages [85]. Headers should contain sequence numbers of regular messages. The possibility of non-FIFO channel is justified in a distributed environment, since it is possible for messages to be routed through different channels and reach the destination out of order.

In a centralized algorithm like Chandy-lamport [22], there is one node which always initiates the checkpoints and coordinates the participating nodes. The disadvantage of a centralized algorithm is that all nodes have to initiate checkpoints whenever the centralized node decides to checkpoint. Nodes can be given autonomy in initiating checkpoints by allowing any node in the system to initiate checkpoints. Such a distributed checkpointing algorithm can initiate complete checkpointing [50] or selective checkpointing [44].

4. RELATED WORK

A survey of the literature on fault tolerant checkpointing shows that a large number of papers have been published. A majority of them have Checkpointing algorithms for parallel and distributed computing been obtained by relaxing many of the assumptions made by Chandy and Lamport (1985); the main aim of improving the earlier extensions of the Chandy & Lamport (1985) algorithms was to minimize the overhead of coordination between processes in a multiprocessor system. Few number of algorithms have been proposed to checkpoint shared-memory multiprocessors and primarily extend cache coherence protocols to maintain a consistent memory. These algorithms assume the main memory to be safe and do not save context in disk. More recently, algorithms have been proposed for distributed shared-memory systems. In these systems also maintenance of cache coherence of the logical global memory is important for checkpoints. As the physical memory is distributed it is necessary to save main memory contents in the disk. Thus context saving overhead is higher when compared to shared-memory systems. We also see that most of the algorithms assume no prior knowledge on the structure of programs meant for execution on multiprocessors. The design of algorithms for distributed systems and their communication costs have been based on the assumptions that the location of hosts in the network do not change and the connectivity amongst the hosts is static in the absence of failures. However, with the emergence of mobile computing, these assumptions are no longer valid. Additionally, mobile hosts have tight constraints on power consumption and bandwidth of the wireless links connecting MHs to their local MSSs is limited.

The Chandy-Lamport [22] algorithm is one of the earliest non-blocking all-process coordinated checkpointing algorithm for static nodes. In this algorithm, markers are sent along all channels in the network which leads to a message complexity of $O(N^2)$, and requires channels to be FIFO. To relax the FIFO assumption, Lai and Yang [50] proposed an algorithm. In this algorithm, when a process takes a checkpoint, it piggybacks a flag to the message it sends out from each channel. The receiver checks the piggybacked flag to see if there is a need to take a checkpoint before processing the message. If so, it takes a checkpoint before processing the message to avoid an inconsistency. To record the channel information, each process needs to maintain the entire message history on each channel as part of the local checkpoint. It requires all processes to take checkpoints. Elnozahy et al. [28] proposed an all-process non-blocking synchronous checkpointing algorithm with a message complexity of O(N). They use checkpoint sequence numbers to identify orphan messages, thus avoiding the need for processes to be blocked during checkpointing. However, this approach requires the initiator to communicate with all processes in the computation. In the algorithm proposed by Silva and Silva [85], the processes which did not communicate with others during the previous checkpointing interval do not need to take new checkpoints. Both these algorithms [28], [85], assume that a distinguished initiator decides when to initiate checkpointing procedure. Therefore, they suffer from the disadvantages of centralized algorithms, such as one-site failure, traffic bottlenecks etc.

Leu and Bhargawa [51] proposed an algorithm which is resilient to multiple process failures and does not assume that the channel is FIFO, which is a requirement in [44]. However, these two algorithms [44], [51] do not consider lost messages in checkpointing and recovery; they assume a sliding window kind of scheme to deal with message loss problem. Dang and Park

[25] proposed an algorithm to address both orphan messages and lost messages.

In paper [15], the first coordinated checkpointing protocol was proposed. It assumes that all communications are atomic, which is too restrictive. Koo-Toeg [44] proposed a minimum-process coordinated checkpointing protocol which relaxes the assumption that all communications are atomic. It reduces the number of synchronization messages and number of checkpoints. The initiator process sends the checkpoint request to P_i only if it has received some m from P_i in the current CI. Similarly, P_i sends the checkpoint request to any process P_j only if P_i has received some m from P_j in the current CI. In this way, a checkpointing tree is formed and at last leaf node processes of the tree take their checkpoints. The time taken to collect coordinated checkpoint in mobile systems may be too large due to mobility, disconnections and unreliable wireless channels. As the processes are blocked during checkpointing and this extensive blocking may degrade the systems performance.

Cao and Singhal [19] proposed minimum-process blocking algorithm for mobile systems. In this algorithm, blocking time is significantly reduced as compared to [44]. Every process maintains its direct dependencies in a bit array of length n for n processes. Initiator process collects the direct dependency vectors of all processes, computes minimum set. After that, it broadcasts the checkpoint request along with the minimum set to all processes. During the period, a process sends its dependency vector to the initiator process and receives the minimum set, it remains in the blocking period. A process takes its checkpoint if it is in the minimum set.

In algorithm [44], if any of the relevant process is not able to take its checkpoint in an initiation, the entire checkpointing process of that particular initiation is aborted. Kim and Park [45] proposed an improved scheme to address failures during checkpointing. It allows the new checkpoints in some subtrees to be committed. In the approach, a process commits its tentative checkpoint if none of the processes, on which it transitively depends, fails; and the consistent recovery line is advanced for those processes that committed their checkpoints. The initiator and other processes which transitively depend on the failed process have to abort their tentative checkpoints. Thus, in case of a node failure during checkpointing, total abort of the checkpointing is avoided.

To further reduce the system messages, needed to synchronize the checkpointing, loosely synchronized clocks are used [23], [63], [79], [84]. Neves et al. [63] gave a loosely synchronized coordinated checkpointing protocol that removes the overhead of synchronization. This approach assumes that the clocks at the processes are loosely synchronized. Loosely synchronized clocks can trigger the local checkpoints at all the processes roughly at the same time without a coordinator. After taking a checkpoint, a process waits for a period, which is sum of maximum time to detect a failure of other process in the system and the maximum deviation between clocks. It is assumed that all checkpoints belonging to a particular coordination session have been taken without the need of exchanging any message. If a failure occurs, it is detected within the specified time and the protocol is aborted. Sinha and Ren [75] devised a tool-assisted method for the formal verification of a timestamp-based checkpointing protocol.

All the above mentioned algorithms strive to reduce the overhead associated with coordinated checkpointing. Efforts are made to reduce the synchronization messages, minimize the number of processes to checkpoint [19], [44] and to make the algorithms non-intrusive [22], [28]. The above mentioned algorithms are either minimum-process or non-intrusive. Prakash and Singhal [72] were first to give minimum-process non-intrusive coordinated checkpointing protocol for mobile distributed systems. But their algorithm may lead to inconsistencies [19]. In [19], it was proved that there does not exist a minimum-process non-intrusive coordinated checkpointing algorithm. Hence, in minimum-process coordinated checkpointing algorithms, some blocking of the processes takes place [19], [44], or some useless checkpoints are taken [20], [48], [64].

In coordinated checkpointing protocols, we may require piggybacking of integer csn (checkpoint sequence number) along with normal messages [20], [21], [28], [64], [48]. L. Kumar et al. [47] proposed an all-process non-intrusive checkpointing protocol for distributed systems, where just one bit is piggybacked along with normal messages. This is done by incurring extra overhead of vector transfers during checkpointing.

Cao and Singhal [20] achieved non-intrusiveness in the minimum-process algorithm by introducing the concept of mutable checkpoints. In their algorithm, initiator, say P_{in}, sends the checkpoint request to any process, say P_j, only if P_{in} receives m from P_j in the current CI. P_j takes its tentative checkpoint if P_j has sent m to P_{in} in the current CI; otherwise, P_j concludes that the checkpoint request is a useless one. Similarly, when P_j takes its tentative checkpoint, it propagates the checkpoint request to other processes. This process is continued till the checkpoint request reaches all the processes on which the initiator transitively depends and a checkpointing tree is formed. During checkpointing, if P_i receives m from P_j such that P_j has taken some checkpoint in the current initiation before sending m, P_i may be forced to take a checkpoint, called mutable checkpoint. If P_i is not in the minimum set, its mutable checkpoint is useless and is discarded on commit. The huge data structure MR[] is also attached with the checkpoint requests to reduce the number of useless checkpoint requests. The response from each process is sent directly to initiator.

The algorithm [20] has been designed to allow its concurrent executions using the technique proposed in [73]. Ni et al [61] have shown that the Cao-Singhal algorithm [20] may lead to inconsistencies during concurrent executions. The authors [61] also updated the algorithm proposed in [20] to allow concurrent executions. The number of useless checkpoints in [20] may be exceedingly high in some situations [48]. L. Kumar et. al [48] and P. Kumar et. al [64] reduced the height of the checkpointing tree and the number of useless checkpoints by keeping non-intrusiveness intact, at the extra cost of maintaining and collecting dependency vectors, computing the minimum set and broadcasting it on the static network along with the checkpoint request. In algorithm [48], after sending the dependency vector

and before receiving the minimum set, P_i processes m received from P_j if any of the following conditions is met:

P_i is directly dependent upon P_j and P_j has not taken any checkpoint for the current initiation before sending m.

P_j has taken some permanent checkpoint after sending m.

P_i has already taken its induced checkpoint for the current initiation

P_i has not sent any message since last committed checkpoint

Otherwise, P_i takes its induced checkpoint [similar to mutable checkpoint] before processing m.

On receiving the minimum set, if P_i finds that it is not a member of the minimum set, it discards its induced checkpoint, if any; otherwise P_i takes its tentative checkpoint or converts its induced checkpoints into tentative one. In this algorithm, no checkpointing tree is formed.

In algorithm [64], on receiving the minimum set, a process takes its tentative checkpoint if it is in the minimum set; otherwise, it ignores the request. When a process P_i takes its tentative checkpoint, it sends the checkpoint request to P_j if P_i is directly dependent upon P_j and P_j is not in the computed minimum set. When P_i receives m from P_j, P_i takes its induced checkpoint before processing m only if following conditions are met: (i) P_j has taken some checkpoint in the current initiation before sending m (ii) P_i has not taken any checkpoint in the current initiation (iii) P_i has sent at least one message since last permanent checkpoint. On commit, if P_i finds that it is not a member of the minimum set, it discards its induced checkpoint, if any. Basically, the algorithms, proposed in [64] and [48], try to minimize the period during which a process may be forced to take its induced/mutable checkpoint. By reducing this period, the number of useless checkpoints is automatically reduced.

Singh and Cabillic [71] proposed a minimum-process non-intrusive coordinated checkpointing protocol for deterministic mobile systems. In the first phase, a process starts checkpointing as checkpoint initiator by sending requests to processes over which it directly depends. When a process receives a checkpoint request, it takes the checkpoint, propagates the checkpoint request to processes over which it directly depends, and continues its processing. During the checkpointing, when P_i sends a computation message to P_j, it also sends id of current checkpoint initiator, if P_i has taken a checkpoint, otherwise a NULL value. When P_j receives the message, it checks the value of checkpoint initiator. If it is not NULL, then P_j knows that checkpointing is going on and it might receive a checkpoint request later; thus it saves the anti-message of the received message before processing it. However, if value of checkpoint initiator is NULL, it simply processes the message. After taking its checkpoint, a process stores the messages in the serial order. An anti-message is also stored for such messages. This storing is done only during the checkpointing to insure the same processing in the recovery phase. In the second phase, checkpoint initiator asks processes to make their checkpoint permanent.

It becomes difficult for multiple MH's to checkpoint synchronously due to disconnections and unreliable wireless channels [34], [35], [36], [60]. MHs are prone to frequent failures, which will require frequent rollback of all processes. Higaki and Takizawa [34] proposed a hybrid checkpointing protocol, where fixed hosts checkpoint synchronously and MHs checkpoint independently. Mobile stations use message logging and checkpointing, while fixed stations use only checkpointing, to form a consistent global state.

Acharya and Badrinath [1] proposed asynchronous checkpointing algorithm for distributed applications on mobile distributed systems. They gave following reasons for neglecting synchronous checkpointing for mobile systems: 1) high cost of locating MHs because in the Chandy Lamport [22] kind of algorithm MH has to receive requests along every incoming link and 2) non-availability of the local checkpoint of an disconnected MH during synchronous checkpointing. In [1], an MH has to take its checkpoint whenever a message reception is preceded by a message transmission at that node. If the transmission and reception messages are interleaved, the number local checkpoints will be equal to half the number of computation messages. This is likely to impose exceedingly high checkpointing overhead.

In uncoordinated algorithms [14], [86], every process may accumulate multiple local checkpoints and logs on the stable storage during normal operation. A checkpoint is discarded if it is determined that it will no longer be needed for recovery. An uncoordinated checkpointing approach is not suitable for mobile systems due to following reasons [20], [72]. If the frequency of local checkpoints is high, each process will have multiple checkpoints, which require a large amount of stable storage and incurs exceedingly high communication overhead. These overheads can be reduced by taking local checkpoints less frequently. However, it will increase the recovery time as greater rollback will be required. Although Some algorithms were proposed to reduce the number of checkpoints to be saved on stable storage, yet, to ensure correctness, a process still needs to keep many more checkpoints in uncoordinated checkpointing algorithms [55], [58], [59], [97]. Generally speaking, uncoordinated checkpointing approaches suffer from the complexities of finding a consistent recovery line after the failure, domino-effect, high stable storage overhead of saving multiple checkpoints of each process, and the overhead of garbage collection. Hence, coordinated checkpointing has many advantages over uncoordinated checkpointing algorithms, especially, for mobile distributed systems. Asynchronous checkpointing with message logging is quite effective for checkpointing mobile systems [34], [69], [98]. In paper [41], a causal message logging protocol for mobile nodes in mobile computing environments has been proposed.

5. REFERENCES

[1] Acharya A. and Badrinath B. R., "Checkpointing Distributed Applications on Mobile Computers," Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems, pp. 73-80, September 1994.

[2] Acharya A., "Structuring Distributed Algorithms and Services for networks with Mobile Hosts", Ph.D. Thesis, Rutgers University, 1995.

[3] Alvisi, Lorenzo and Marzullo, Keith," Message Logging: Pessimistic, Optimistic, Causal, and Optimal", IEEE Transactions on Software Engineering, Vol. 24, No. 2, February 1998, pp. 149-159.

[4] L. Alvisi, Hoppe, B., Marzullo, K., "Nonblocking and Orphan-Free message Logging Protocol," Proc. of 23rd Fault Tolerant Computing Symp., pp. 145-154, June 1993.

[5] L. Alvisi," Understanding the Message Logging Paradigm for Masking Process Crashes," Ph.D. Thesis, Cornell Univ., Dept. of Computer Science, Jan. 1996. Available as Technical Report TR-96-1577.

[6] L. Alvisi and K. Marzullo," Tradeoffs in implementing Optimal Message Logging Protocol", Proc. 15th Symp. Principles of Distributed Computing, pp. 58-67, ACM, June, 1996.

[7] Adnan Agbaria, Wiilliam H Sanders," Distributed Snapshots for Mobile Computing Systems", IEEE Intl. Conf. PERCOM'04, pp. 1-10, 2004.

[8] Avi Ziv and Jehoshua Bruck, " Checkpointing in Parallel and Distributed Systems", Book Chapter from Parallel and Distributed Computing Handbook edited by Albert Z. H. Zomaya, pp. 274-302, Mc Graw Hill, 1996.

[9] A. Borg, J. Baumbach, and S. Glazer," A Message System Supporting Fault Tolerance", Proc. Symp. Operating System Principles, pp. 90-99, ACM SIG OPS, Oct. 1983.

[10] Adnan Agbaria, William H. Sanders, " Distributed Snapshots for Mobile Computing Systems", Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications (Percom'04), pp. 1-10, 2004.

[11] Baldoni R., Hélary J-M., Mostefaoui A. and Raynal M., " Rollback Dependency Trackability: A Minimial Characterization and its Protocol", Information and Computation, 165, pp. 144-173, 2003.

[12] Baldoni R., Hélary J-M., Mostefaoui A. and Raynal M., "A Communication- Induced Checkpointing Protocol that Ensures Rollback-Dependency Trackability," Proceedings of the International Symposium on Fault-Tolerant-Computing Systems, pp. 68-77, June 1997.

[13] Bhagwat P., and Perkins, C.E., "A mobile Networking System based on Internet Protocol (IP)",USENIX Symposium on Mobile and Location-Independent Computing, August 1993.

[14] Bhargava B. and Lian S. R., "Independent Checkpointing and Concurrent Rollback for Recovery in Distributed Systems-An Optimistic Approach," Proceedings of 17th IEEE Symposium on Reliable Distributed Systems, pp. 3-12, 1988.

[15] G. Barigazzi and L. Strigni, " Application-Transparent Setting of Recovery Points", Digest of Papers Fault Tolerant Computing Systems-13, pp. 48-55, 1983.

[16] Badrinath B. R, Acharya A., T. Imielinski "Structuring Distributed Algorithms for Mobile Hosts", Proc. 14th Int. Conf. Distributed Computing Systems, June 1994.

[17] Badrinath B. R, Acharya A., T. Imielinski " Designing Distributed Algorithms for Mobile Computing Networks", Computer Communications, Vol. 19, No. 4, 1996.

[18] Cao G. and Singhal M., "On coordinated checkpointing in Distributed Systems", IEEE Transactions on Parallel and Distributed Systems, vol. 9, no.12, pp. 1213-1225, Dec 1998.

[19] Cao G. and Singhal M., "On the Impossibility of Min-process Non-blocking Checkpointing and an Efficient Checkpointing Algorithm for Mobile Computing Systems," Proceedings of International Conference on Parallel Processing, pp. 37-44, August 1998.

[20] Cao G. and Singhal M., "Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing systems," IEEE Transaction On Parallel and Distributed Systems, vol. 12, no. 2, pp. 157-172, February 2001.

[21] Cao G. and Singhal M., "Checkpointing with Mutable Checkpoints", Theoretical Computer Science, 290(2003), pp. 1127-1148.

[22] Chandy K. M. and Lamport L., "Distributed Snapshots: Determining Global State of Distributed Systems," ACM Transaction on Computing Systems, vol. 3, No. 1, pp. 63-75, February 1985.

[23] F. Cristian and F. Jahanian, " A timestamp-based Checkpointing Protocol for Long Lived Distributed Computations", Proc IEEE Symp. Reliable Distributed Systems, pp. 12-20, 1991.

[24] David R. Jefferson, "Virtual Time", ACM Transactions on Programming Languages and Systems, Vol. 7, NO.3, pp 404-425, July 1985.

[25] Dang Y., Park, E.K. ," Checkpointing and Rollback-Recovery Algorithms in Distributed Systems", Journal of Systems and Software, pp. 59-71, April 1994.

[26] Dieter Kranzlmuller, Nam Thoai, Jens Volkert," Error Detection in Large Scale Parallel Programs with Long runtimes, Future Generation Computer Systems 19, pp. 689-700, 2003.

[27] Elnozahy E.N., Alvisi L., Wang Y.M. and Johnson D.B., "A Survey of Rollback-Recovery Protocols in Message-Passing Systems," ACM Computing Surveys, vol. 34, no. 3, pp. 375-408, 2002.

[28] Elnozahy E.N., Johnson D.B. and Zwaenepoel W., "The Performance of Consistent Checkpointing," Proceedings of the 11th Symposium on Reliable Distributed Systems, pp. 39-47, October 1992.

[29] Elnozahy and Zwaenepoel W, " Manetho: Transparent Roll-back Recovery with Low-overhead, Limited Rollback and Fast Output Commit," IEEE Trans. Computers, vol. 41, no. 5, pp. 526-531, May 1992.

[30] Elnozahy and Zwaenepoel W, " On the Use and Implementation of Message Logging," 24th int'l Symp. Fault Tolerant Computing, pp. 298-307, IEEE Computer Society, June 1994.

[31] George H. Forman and John Zahorjan, "The Challenges of Mobile Computing", IEEE Computers vol. 27, no. 4, April 1994, pp. 38-47.

[32] Richard C. Gass and Bidyut Gupta," An Efficient Checkpointing Scheme for Mobile Computing Systems", European Simulation Symposium, Oct 18-20, 2001, pp. 1-6.

[33] Hélary J. M., Mostefaoui A. and Raynal M., "Communication-Induced Determination of Consistent Snapshots," Proceedings of the 28th International Symposium on Fault-Tolerant Computing, pp. 208-217, June 1998.

[34] Higaki H. and Takizawa M., "Checkpoint-recovery Protocol for Reliable Mobile Systems," Trans. of Information processing Japan, vol. 40, no.1, pp. 236-244, Jan. 1999.

[35] Higaki H. and Takizawa M., "Recovery Protocol for Mobile Checkpointing", IEEE 9th International Conference on Database Expert Systems Applications, Viena, pp. 520-525, 1998

[36] Higaki H. and Takizawa M., "Checkpoint Recovery Protocol for Reliable Mobile Systems", 17th Symposium on Reliable Distributed Systems, pp. 93-99, Oct. 1998.

[37] Ioannidis, J., Duchamp, D., and Maguire, G.Q., "IP-based protocols for Mobile Internetworking", In Proc. of ACM SIGCOMM Symposium on Communications, Architectures, and Protocols, pp. 235-245, September 1991.

[38] Johnson, D.B., Zwaenepoel, W., "Sender-based message logging", In Proceedingss of 17th international Symposium on Fault-Tolerant Computing, pp 14-19, 1987.

[39] Johnson, D.B., Zwaenepoel, W., "Recovery in Distributed Systems using optimistic message logging and checkpointing. In 7th ACM Symposium on Principles of Distributed Computing, pp 171-181, 1988.

[40] D. Johnson, "Distributed System Fault Tolerance Using Message Logging and Checkpointing," Ph.D. Thesis, Rice Univ., Dec. 1989.

[41] JinHo Ahn, Sung-Gi Min, Chong-Sun Hwang, "A Causal Message Logging Protocol for Mobile Nodes in Mobile Computing Environments", Future Generation Computer Systems 20, pp 663-686, 2004.

[42] Kalaiselvi, S., Rajaraman, V., "A Survey of Checkpointing Algorithms for Parallel and Distributed Systems", Sadhna, Vol. 25, Part 5, October 2000, pp. 489-510.

[43] Kistler, J., and Satyanaranyan, M., " Disconnected Operation in the Coda file system", ACM Trans. on Computer Systems 10, 1 (Feb. 1992).

[44] Koo R. and Toueg S., "Checkpointing and Roll-Back Recovery for Distributed Systems," IEEE Trans. on Software Engineering, vol. 13, no. 1, pp. 23-31, January 1987.

[45] J.L. Kim, T. Park, "An efficient Protocol for checkpointing Recovery in Distributed Systems," IEEE Trans. Parallel and Distributed Systems, pp. 955-960, Aug. 1993.

[46] Kyne-Sup BYUN, Sung_Hwa LIM, Jai-Hoon KIM," Two-Tier Checkpointing Algorithm Using MSS in Wireless Networks", IEICE Trans. Communications, Vol E86-B, No. 7, pp. 2136-2142, July 2003.

[47] L. Kumar, M. Misra, R.C. Joshi, "Checkpointing in Distributed Computing Systems" Book Chapter "Concurrency in Dependable Computing", pp. 273-92, 2002.

[48] L. Kumar, M. Misra, R.C. Joshi, "Low overhead optimal checkpointing for mobile distributed systems" Proceedings. 19th IEEE International Conference on Data Engineering, pp 686 – 88, 2003.

[49] Lalit Kumar, Parveen Kumar, R K Chauhan, "Logging based Coordinated Checkpointing in Mobile Distributed Computing Systems", IETE Journal of Research, vol. 51, no. 6, pp. 485-490, 2005.

[50] T.H. Lai and T.H. Yang," On Distributed Snapshots", Information Processing Letters, vol. 25, pp. 153-158, 1987.

[51] P.J. Leu and B.Bhargawa, " Concurrent Robust Checkpointing and Recovery in Distributed Systems", Proceeding Fourth Intl Conf. Data Engg. Pp. 154-163, Feb. 1988.

[52] L. Lamport, "Time, clocks and ordering of events in a distributed system" Comm. ACM, vol.21, no.7, pp. 558-565, July 1978.

[53] Lalit Kumar, Parveen Kumar, R K Chauhan, "Pitfalls in Minimum-process Coordinated Checkpointing protocols for Mobile Distributed", ACCST Journal of Research, Volume III, No. 1, 2005 pp. 51-56.

[54] Lalit Kumar, Parveen Kumar, R K Chauhan, "Message Logging and Checkpointing in Mobile Computing", Journal of Multi-disciplinary Engineering Technologies, Vol.1, No.1, 2005, pp. 61-66.

[55] Manivannan D. and Singhal M., "Quasi-Synchronous Checkpointing: Models, Characterization, and Classification," IEEE Trans. Parallel and Distributed Systems, vol. 10, no. 7, pp. 703-713, July 1999.

[56] Manivannan D., Netzer R. H. and Singhal M., "Finding Consistent Global Checkpoints in a Distributed Computation," IEEE Transactions on Parallel & Distributed Systems, vol. 8, no. 6, pp. 623-627, June 1997.

[57] Yoshifumi Manabe," A Distributed Consistent Global Checkpoint Algorithm for Distributed Mobile Systems", 8th Int'l Conference on Parallel and Distributed Systems", pp. 125-132, 2001.

[58] Mannivannam, D., Singhal, M., "Failure Recovery based on Quasi-Synchronous Checkpointing in Mobile Computing Systems", In TR No. OSU-CISRC-7/96-TR-36, Dept of Computer and Information Science, The Ohio State University, 1996.

[59] Mannivannam, D., Singhal, M., " A Low overhead Recovery Techniques using Quasi Synchronous Checkpointing", Proc. 16th int'l conf. Distributed Computing Systems, pp 100-107, May 1996.

[60] Yoshinori Morita, Kengo Hiraga and Hiroaki Higaki," Hybrid Checkpoint Protocol for Supporting Mobile-to-

Mobile Communication", Proc. Of the International Conference on Information Networking, 2001.

[61] Ni, W., S. Vrbsky and S. Ray, "Pitfalls in Distributed Nonblocking Checkpointing", Journal of Interconnection Networks, Vol. 1 No. 5, pp. 47-78, March 2004.

[62] Netzer, R.H. and Xu,J ,"Necessary and Sufficient Conditions for Consistent Global Snapshots", IEEE Trans. Parallel and Distributed Systems 6,2, pp 165-169, 1995.

[63] Neves N. and Fuchs W. K., "Adaptive Recovery for Mobile Environments," Communications of the ACM, vol. 40, no. 1, pp. 68-74, January 1997.

[64] Parveen Kumar, Lalit Kumar, R K Chauhan, V K Gupta "A Non-Intrusive Minimum Process Synchronous Checkpointing Protocol for Mobile Distributed Systems" Proceedings of IEEE ICPWC-2005, January 2005.

[65] Parveen Kumar, Lalit Kumar, R K Chauhan, "A low overhead Non-intrusive Hybrid Synchronous checkpointing protocol for mobile systems", Journal of Multidisciplinary Engineering Technologies, Vol.1, No. 1, pp 40-50, 2005.

[66] Parveen Kumar, Lalit Kumar, R K Chauhan, "Synchronous Checkpointing Protocols for Mobile Distributed Systems: A Comparative Study", International Journal of information and computing science, Volume 8, No.2, 2005, pp 14-21.

[67] Parveen Kumar, Lalit Kumar, R K Chauhan, "A Hybrid Coordinated Checkpointing Protocol for Mobile Computing Systems", IETE journal of research, Vol 52, No. 2&3, pp 247-254, 2006.

[68] Parveen Kumar, Lalit Kumar, R K Chauhan, "A Synchronous Checkpointing Protocol for Mobile Distributed Systems: A Probabilistic Approach, Accepted for Publication in International Journal of Information and Computer Security.

[69] Pradhan D.K., Krishana P.P. and Vaidya N.H., "Recoverable Mobile Environment: Design and Trade-off Analysis," Proceedings 26th International Symposium on Fault-Tolerant Computing, pp. 16-25, 1996.

[70] Pradhan D.K. and Vaidya N., "Roll-forward Checkpointing Scheme: Concurrent Retry with Non-dedicated Spares," Proceedings of the IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems, pp. 166-174, July 1992.

[71] Pushpendra Singh, Gilbert Cabillic, "A Checkpointing Algorithm for Mobile Computing Environment", LNCS, No. 2775, pp 65-74, 2003.

[72] Prakash R. and Singhal M., "Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems," IEEE Transaction On Parallel and Distributed Systems, vol. 7, no. 10, pp. 1035-1048, October1996.

[73] Prakash R. and Singhal M., "Maximum Global Snapshot with Concurrent Initiations", Proc. Sixth IEEE Symp. Parallel and Distributed Processing, pp. 344-51, Oct. 1994.

[74] M.L. Powell and D.L. Presotto, "Publishing: A Reliable Broadcase Communication Mechanism", Proc. ninth Symp. Operating System Principles, pp. 100-109, ACM SIGOPS, Oct. 1983.

[75] Purnendu Sinha, Da Qi Ren, "Formal Verification of Dependable Distributed Protocols", Information and Software Technology, 45, pp. 873-888, 2003.

[76] Quagila, F., Ciciani, R., Baldoni, R., " Checkpointing Protocols in Distributed Systems with Mobile Hosts: A Performance Analysis", IPPS/SPDP Workshop, pp. 742-755, 1998.

[77] Randall, B, " System Structure for Software Fault Tolerance", IEEE Trans. on Software Engineering, 1,2, 220-232, 1975.

[78] Russell, D.L., "State Restoration in Systems of Communicating Processes", IEEE Trans. Software Engineering, 6,2. 183-194, 1980.

[79] Ramanathan, P. and K.G. Shin, "Use of Common Time Base for Checkpointing and Rollback Recovery in a Distributed System", IEEE Trans. Software Engg., pp. 571-583, June 1993.

[80] R K Chauhan, Parveen Kumar, Lalit Kumar, "A coordinated checkpointing protocol for mobile computing systems", International Journal of information and computing science, Accepted for Publication, Vol 9, No. 1, 2006.

[81] R K Chauhan, Parveen Kumar, Lalit Kumar, "Hybrid and intrusive synchronous checkpointing protocols for mobile distributed systems", Accepted for publication in ACCST Journal of Research, Volume IV, No. 4, 2006

[82] R K Chauhan, Parveen Kumar, Lalit Kumar, "Non-intrusive Coordinated Checkpointing Protocols for Mobile Computing Systems : A Critical Survey, ACCST Journal of Research, to be published in Volume IV, No. 3, 2006.

[83] R K Chauhan, Parveen Kumar, Lalit Kumar, "Checkpointing Distributed Applications on Mobile Computers", Journal of Multidisciplinary Engineering and Technologies, Vol. 2 No.1, Jan. 2006.

[84] Ssu K.F., Yao B., Fuchs W.K. and Neves N. F., "Adaptive Checkpointing with Storage Management for Mobile Environments," IEEE Transactions on Reliability, vol. 48, no. 4, pp. 315-324, December 1999.

[85] Silva, L.M. and J.G. Silva, "Global checkpointing for distributed programs", Proc. 11th symp. Reliable Distributed Systems, pp. 155-62, Oct. 1992.

[86] Storm R., and Temini, S., "Optimistic Recovery in Distributed Systems", ACM Trans. Computer Systems, Aug, 1985, pp. 204-226.

[87] A.P. Sistla and J.L. Welch," Efficient Distributed Recovery Using Message Logging", Proc. 18th Symp. Principles of Distributed Computing", pp 223-238, Aug. 1989.

[88] Tamir, Y., Sequin, C.H., "Error Recovery in multi-computers using global checkpoints", In Proceedings of the International Conference on Parallel Processing, pp. 32-41, 1984.

[89] Terakota, F., Yokote, Y., and Tokoro, M., "A Network Architecture providing host migration transparency", Proc, of ACM SIGCOMM 91, September 1991.

[90] S. Venketasan and T.Y. Juang, "Efficient Algorithms for Optimistic Crash recovery", Distributed Computing, vol. 8, no. 2, pp. 105-114, June 1994.

[91] S. Venketasan, "Message-Optimal Incremental Snapshots", Computer and Software Engineering, vol.1, no.3, pp. 211-231, 1993.

[92] S. Venketasan, " Optimistic Crash recovery Without Rolling back Non-Faulty Processors", Information Sciences, 1993.

[93] S. Venketasan and T.T.Y. Juang, "Low Overhead optimistic crash Recovery", Proc. 11th Int. Conf. Distributed Computing systems, pp. 454-461, 1991.

[94] Wada H., Yozawa, T., Ohnishi, T. and Tanaka, Y., "Mobile Computing Environment based on internet packet forwarding", Winter Usenix, Jan. 1993.

[95] Wang Y. M., Huang Y., Vo K.P., Chung P.Y. and Kintala C., "Checkpointing and its Applications," Proceedings of the 25th International Symposium on Fault-Tolerant Computing (FTCS-25),pp. 22-31, June 1995.

[96] Wood, W.G., "A Decentralized Recovery Control Protocol", 1981 IEEE Symposium on Fault Tolerant Computing, 1981.

[97] Wang Y. and Fuchs, W.K., "Lazy Checkpoint Coordination for Bounding Rollback Propagation," Proc. 12th Symp. Reliable Distributed Systems, pp. 78-85, Oct. 1993.

[98] Bin Yao, Kuo-Feng Ssu & W. Kect Fuchs, "Message Logging in Mobile Computing", Proceedings of international conference on FTCS, pp 294-301, 1999.

[99] Yasuro Sato, Michiko Inoue, Toshimitsu Masuzawa, Hideo Fujiwara, " A Snapshot Algorithm for Distributed Mobile Systems" Proceedings of the 16th ICDCS, pp734-743,1996.